lme4 extras

Ben Bolker

May 4, 2012

Contents

This vignette is intended to document some extra tricks that can be used with `lme4` models. Some of them are included here because they are statistically non-rigorous and we didn't want to build them into automatic functions that could be applied unthinkingly, but recipes are supplied here for use **at your own risk** and assuming that you know what you're doing ...

1 To do

In principle, we should be able to get confidence intervals on parameters *and* confidence intervals on predictions via (1) quadratic/Wald approximation (ignoring uncertainty of θ, and possibly of u, for CIs of prediction); (2) cheesy MCMC (in this case, for LMMs, we need a way to retrieve new values of sigma and the fixed effects conditional on θ); (3) parametric bootstrap.

The basic machinery for this is (1) functions for converting among parameterizations of the random effects, i.e. from a (standard deviation, correlation) vectors to θ (concatenated Cholesky-factor vector); (2) a way to extract a deviance function from a fit (i.e. `mkdevfun`) and (3) a way to simulate values from a fit (i.e. `simulate`), along with basic components (matrix inversion, etc.).

We'll see how far I get.

2 Fit basic models

In this section we simply fit a few basic models to use as examples later on.

```
library(lme4)
fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy)
gm1 <- glmer(cbind(incidence, size - incidence) ~
             period + (1 | herd),
             data = cbpp, family = binomial)
```

3 Quadratic confidence intervals on random effects parameters

Extract the deviance function and the ML (or REML) parameters:

```
fm1Fun <- update(fm1,devFunOnly=TRUE)
fm1_thpar <- getME(fm1,"theta")
```

Extract internal functions for converting (standard deviation, correlation) vectors to θ (concatenated Cholesky-factor) parameterization, and vice versa (**this is temporary, until we finalize the definitions/names of these functions ...**)

```
Sv_to_Cv <- lme4:::Sv_to_Cv
Cv_to_Sv <- lme4:::Cv_to_Sv
```

```
fm1FunS <- function(spar) {
    thpar <- Sv_to_Cv(c(spar,NA),s=fm1_spar[4])
    fm1Fun(thpar)
}
```

Use the `numDeriv` package to compute the Hessian (second derivative) matrix at the MLE:

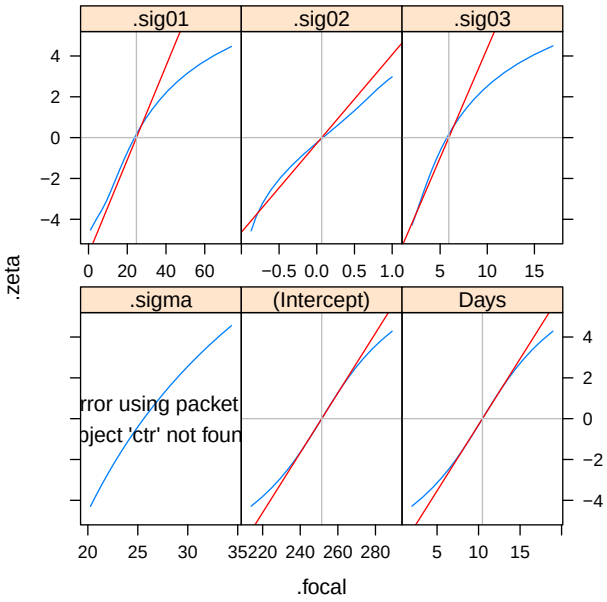
```
library(numDeriv)
fm1_spar <- Cv_to_Sv(fm1_thpar,s=sigma(fm1))
all(abs(Sv_to_Cv(fm1_spar,s=fm1_spar[4])-fm1_thpar)<1e-6)

## [1] TRUE

h <- hessian(fm1FunS,fm1_spar[-4])
```

Variance-covariance matrix of the random-effects (Cholesky) parameters:

```
vcov_ran <- solve(h)
```

Compare profiles to their quadratic approximations, and profile confidence intervals to these approximate (Wald) confidence intervals:

```
pp <- profile(fm1)
```



FIXME: These slopes are not quite right. Why? (Are the profiles correct?) Should we be recomputing σ for each set of θ values? Think about this sometime when I have a brain ...

```
(ci_prof <- confint(pp))
```

```
##                 2.5 %  97.5 %
## .sig01       14.3815  37.716
## .sig02       -0.4815   0.685
## .sig03        3.8012   8.753
## .sigma       22.8983  28.858
## (Intercept) 237.6807 265.130
## Days          7.3587  13.576
```

```
c(fm1_spar,fixef(fm1))+

1.96*outer(c(sqrt(diag(vcov_ran)),NA,sqrt(diag(as.matrix(vcov(fm1))))),
               c(-1,1))
```

```
##                  [,1]     [,2]
##               16.2243  33.257
##               -0.3849   0.516
##                4.0969   7.747
##                    NA      NA
## (Intercept) 238.0289 264.781
## Days          7.4375  13.497
```

4 Approximate confidence intervals on predictions

5 Poor man's MCMC

```
library(MCMCpack)
```

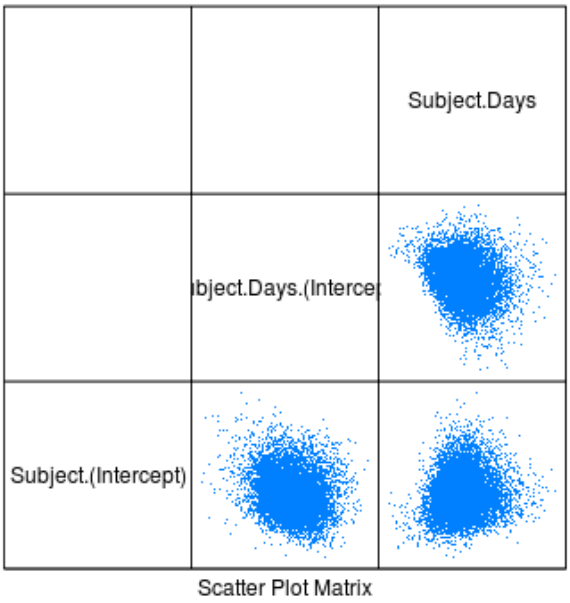
MCMCpack expects a function that gives a value proportional to the log posterior density for any specified set of parameters. We can get lme4 to give us a function for the deviance (by using devFunOnly=TRUE. If we assume all-improper priors (i.e. flat on the scale on which we have defined the parameters), then the log posterior density is $-D/2$:

```
fm1_metropfun <- function(x) {
  ## getME(.,"lower")?
  if (any(x<fm1@lower)) -Inf else -fm1Fun(x)/2
}
```

```
fm1_mcmc_out <- MCMCmetrop1R(fm1_metropfun,fm1_thpar,seed=12345)

##
##
## @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
## The Metropolis acceptance rate was 0.49678
## @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

colnames(fm1_mcmc_out) <- names(fm1_thpar)
```

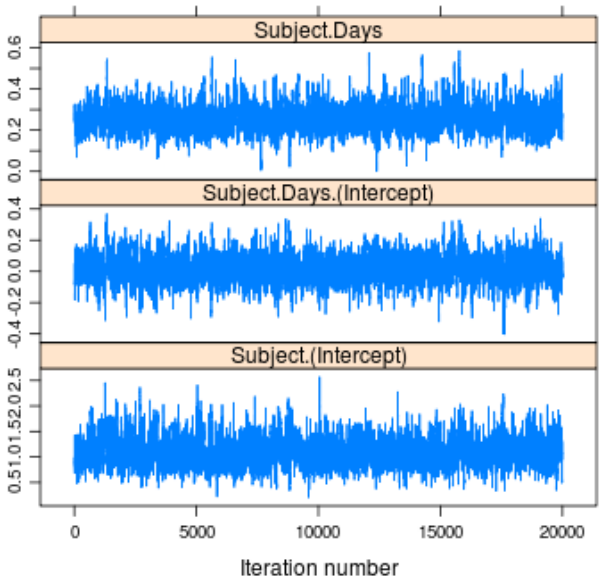
```
library(coda)
```

```
splom(fm1_mcmc_out)
```

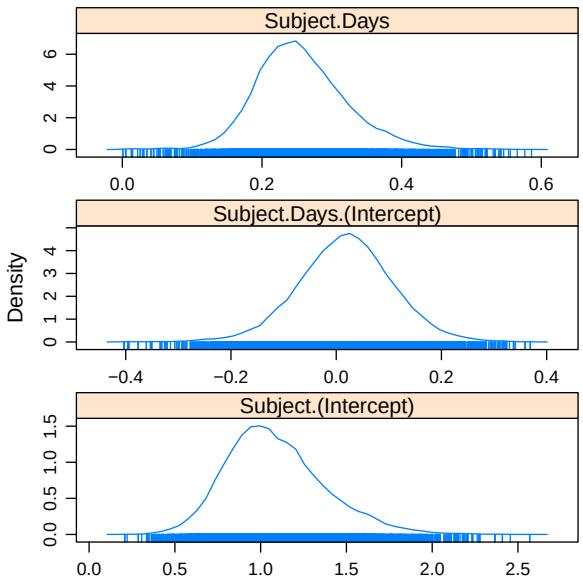


Scatter Plot Matrix

```
xyplot(fm1_mcmc_out)
```



```
densityplot(fm1_mcmc_out,layout=c(1,3))
```




```
HPDinterval(fm1_mcmc_out)

##                              lower  upper
## Subject.(Intercept)        0.5646 1.6712
## Subject.Days.(Intercept) -0.1693 0.1903
## Subject.Days               0.1416 0.3966
## attr(,"Probability")
## [1] 0.95
```

FIXME: if we want to get this on the sd/corr scale we have to figure out how to recalculate sigma for each set of θ values ... for now, just use a fixed sigma

```
sdmat <- as.mcmc(t(apply(fm1_mcmc_out,1,
                         function(x)
c(Cv_to_Sv(x,s=sigma(fm1))))))
```

Highest posterior density intervals:

```
HPDinterval(sdmat)

##      lower   upper
```

```
## 11 14.4493 42.7685
## 12 -0.5457  0.6567
## 13  4.0532 10.5283
##    25.5918 25.5918
## attr(,"Probability")
## [1] 0.95
```

Or quantile-based estimates:

```
t(apply(sdmat,2,quantile,c(0.025,0.975)))
```

```
##       2.5%   97.5%
## 11 15.4823 44.2549
## 12 -0.5194  0.6922
## 13  4.2457 10.8785
##    25.5918 25.5918
```

The latter *should* be translation-invariant, and hence (???) the same as:

```
qq <- t(apply(fm1_mcmc_out,2,quantile,c(0.025,0.975)))
apply(qq,2,function(x) VC(fm1,theta=x,format="sdcorvec"))
```

```
##                               2.5%   97.5%
## Subject.(Intercept)      15.4823 44.2549
## Subject.Days.(Intercept) -0.7561  0.4393
## Subject.Days              5.6209 11.4225
```

(Only true for variable 1, although not terribly different: think about this (i.e. the effect of $\theta\theta^T$) some more ...)

If we have the Cholesky form

$$\begin{pmatrix} c_1 & 0 \\ c_2 & c_3 \end{pmatrix}$$

and take the cross-product, we get

$$\begin{pmatrix} c_1^2 & c_1 c_2 \\ c_1 c_2 & c_2^2 + c_3^2 \end{pmatrix}$$

so it's natural that only element 1 scales as we would expect: all the other terms are not just scale translations of a single element, but combinations of multiple elements.

Should work for GLMMs as well:

```
gm1Fun <- update(gm1,devFunOnly=TRUE)
gm1_par <- c(getME(gm1,"theta"),fixef(gm1))
nt <- length(getME(gm1,"theta"))
```

```
gm1_metropfun <- function(x) {
  if (any(x[seq(nt)]<gm1@lower)) -Inf else -gm1Fun(x)/2
}
```

```
gm1_mcmc_out <- MCMCmetrop1R(gm1_metropfun,gm1_par)

## Error: PIRLS step failed
```

```
colnames(gm1_mcmc_out) <- names(gm1_par)

## Error: object 'gm1_mcmc_out' not found

xyplot(gm1_mcmc_out[,1:2])

## Error: object 'gm1_mcmc_out' not found

splom(gm1_mcmc_out)

## Error: object 'gm1_mcmc_out' not found

HPDinterval(gm1_mcmc_out)

## Error: object 'gm1_mcmc_out' not found
```

6 Confidence intervals on predictions etc. via parametric bootstrap

FIXME: do we want a `as.data.frame.boot` function to retrieve stuff from `bootMer` output?

7 Zero-inflation via the EM algorithm

Adapted from code by Mihoko Minami and Cleridy Lennert (ref??)

```
zipme <- function(cformula, zformula, cfamily=poisson,
                  data, maxitr=20, tol=1e-6, verbose=TRUE) {
  ##   y is the observation from the distribution:
  ##           P(Y=0)=p+(1-p)F(0,lambda)
  ##           P(Y=k)=(1-p)F(k,lambda).
  ##  zformula: formula for logistic regression on
zero-inflation: LHS should be z~
```

```
  ##  cformula: formula for Poisson or NB regression. LHS should
be: y~
  ##   maxitr  : maximum number of iterations
  ##   tol: convergence tolerance
  ##

  m<-nrow(data)
  rname <- as.character(cformula)[2]
  ## initialize z and probz (z=1 -> perfect state; probz is
probability of 0 in imperfect state for poisson)

  z<-numeric(m)
  probz<-numeric(m)
  z[data[[rname]]==0]<- 1/(1+exp(-1))  ## starting value

  ## n.b. we are looking for [3] since zformula has a LHS
  randz <-
length(grep("\\(.*\\|.*\\)",as.character(zformula)[3]))>0
  ## delta is used to gauge convergence. after initialization, it
is the abs. difference between current z and new z.
  itr <- 1
  delta <- 2
  deltainfo <- numeric(maxitr)
  while(delta>tol & itr <= maxitr){
    if (verbose) cat("itr:",itr,"\n")
    ## make (update) working data frame
    bydataw <- data.frame(z=z,data)
    ##
    ## Maximization 1: logistic
    old.z<-z
    if (randz) {
      uu <- glmer(zformula, family=binomial, data=bydataw)
    } else {
      uu <- glm(zformula, family=binomial, data=bydataw)
    }
    ## save current logistic model output
    u <- fitted(uu)
    ##
    ## Maximization 2: poisson/NB loglinear with weights
    vv <- glmer(cformula, family=cfamily, weights=(1-z),
data=bydataw)
    ## save Poisson model output
    v <- fitted(vv)
    ##
    ## Expectation: used to update z with conditional
```

```
expectation;only need to update at y=0.
    zdat <- data[[rname]]  0
    z[zdat] <- u[zdat]/( u[zdat]+(1-u[zdat])*exp(-v[zdat]))
    new.z<-z
    ## updated convergence indicator
    delta<-max(abs(old.z-new.z))
    ## save delta for this iteration; to be output
    deltainfo[itr] <- delta
    itr <- itr+1
  }
  L <- list("zfit"=uu, "cfit"=vv, itr=itr, deltainfo=deltainfo,
z=z)
  ##    uu.binom : output object of logistic regression;
  ##    vv.flm   : output object of poisson regression
  class(L) <- "zipme"
  L
}
```